# PROJECT REPORT

## Project Title: SUDOKU QUEST

## Team members:

**Lavesh Kumar Srivastav (145320004) [Team leader]**

**Sayandeep Roy (145060019)**

**Dipayan Mukherjee (145320014)**

## Project Description:

### ❖ Aim of Project:

The game Sudoku, since its first appearance in 1970's, has been one of the most popular puzzle games of all times. It is often seen that some people have a real knack of solving the puzzle easily, while there are others who face difficulties at first and require more and more practice to acquire that skill.

We have set the goals of our project to aid those who are immensely fond of the game but with amateur skills. The program will work both as a ***Sudoku generator*** by providing solvable Sudoku puzzles for user to play and practice (according to user specified difficulty levels) and also as an intelligent ***Sudoku auto-solver***, which can help the user play the game by providing appropriate 'hints' and 'check' options during play or show the end solution of puzzle if user wants. User will also have the option to give any unsolved puzzle as input to the program and check its solution.

### ❖ Project Ideas:

The project involves basically three broad components:

1) Solving an user defined puzzle taken as inputs by **Back-tracking method**;
2) Generating a puzzle for the user to play;
3) Building the graphics and interfaces of the game using **GTK graphics package.**

### ❖ Project Execution:

#### ➢ Header files to be included:

**1)** gtk/gtk.h; **2)** string.h; **3)** iostream; **4)** stdlib.h; **5)** stdio.h; **6)** gdk/gdk.h; **7)** time.h; **8)** cstdio

## ➢ Functions defined:

- **Sudoku solver program:** **1)** isUnique(); **2)** solveSudoku(); **3)** findUnassignedLocation(); **4)** isSafetoPut(); **5)** checkRow(); **6)** checkBox(); **7)** checkColumn(); **8)** isEqual()
- **Sudoku generator program:** **1)** sudoku_game(); **2)** hint(); **3)** check(); **4)** check1()
- **Graphics and interfaces:** **1)** get_grid(); **2)** menu_response(); **3)** sudoku_game(); **4)** developers(); **5)** difficulty(); **6)** invalid_entry(); **7)** Sudoku_display

## ❖ Scope of the Project:

The program that will create a very efficient automatic Sudoku solver that can solve any Sudoku puzzle given by the user. It can detect whether a puzzle will have a unique solution or not and therefore decide the validity of the puzzle. The program can also provide puzzles for the user to play and offer assistance with 'hint' and 'check' options during gameplay.

## ❖ Backtracking Algorithm:

**The Algorithm that we have chosen to solve a random Sudoku puzzle is called the *Back-tracking algorithm*.** The logic behind this is to basically iterate all the possible solutions for a given unassigned cell and assign them one by one until the Sudoku is solved and 'back-track' whenever required. The summary of steps are given below:

- First in the given Sudoku puzzle, we have to locate an unassigned (empty) cell.
- Then numbers from 1 to 9 are successively checked until a 'valid' number is found i.e. the number should be unique in that row, column and box (3X3) to which the cell belongs.
- This valid number is then assigned to that cell.
- Next, another unassigned location is sought for and this process is repeated to fill that up also.
- The 'back-tracking' comes to action if all the possible numbers from 1 to 9 are invalid for any cell that is currently being checked. The algorithm then moves back to the previous cell and changes that cell's value to another valid number, i.e. back tracks. Afterwards, it moves again to the next cell and the whole process repeats itself.

- A Sudoku will be considered unsolvable if back tracking has been done up to the first filled cell and all digits have been tried and nothing worked even at that cell.

(**Reference:** http://see.stanford.edu/materials/icspacs106b/H19RecBacktrackExamples.pdf)

## ❖ Generating Sudoku Puzzle:

File handling system was involved to store ten valid Sudoku puzzles for each of the difficulty levels. In the .txt file we stored the numbers in such a way that for a single difficulty level, e.g. Beginner level there are a total of 810 numbers, where each set of 81 numbers representing a valid Sudoku. The blank spaces are denoted by the number '0'. Each set is separated by '|' symbol and thus for one difficulty level, there are 10 such sets. Now for the next difficulty level, say 'Amateur', there are also ten such sets of numbers and the last set of numbers in beginner level being separated from the 1st set of Amateur level by the symbol '@'. Same thing is done for the rest of the difficulty levels as well in that ordered manner.

Here is a simple layout:

Beginner: 1st set |Beginner: 2nd set| Beginner: 3rd set|…..Beginner: 10th set @ Amateur: 1st set| Amateur: 2nd set| Amateur: 3rd set|…….Amateur: 10th set @ Expert: 1st set…………………..Veteran: 10th set

**Total no of characters:** 3240 digits + 360 "|" s + 3 "@" s = 3603
**"@" s are at the positions:** 820th , 1640th , 2460th positions.

3) Next, in the generator function, a random number is chosen by using **'rand'** function. For beginner level, the number is chosen from 0 to 9, with each representing one set of 81 numbers. For example, if the random number came out to be 3, then the 81 numbers, starting from the position (3X82) or 246th to (246+81)th position are taken to be displayed in the grid.

4) Now according to the particular set chosen, the numbers from that set are taken up and grid is filled with the numbers and Sudoku is generated. The empty boxes are filled with zero as usual. The Sudoku is displayed on the screen for the user to play.